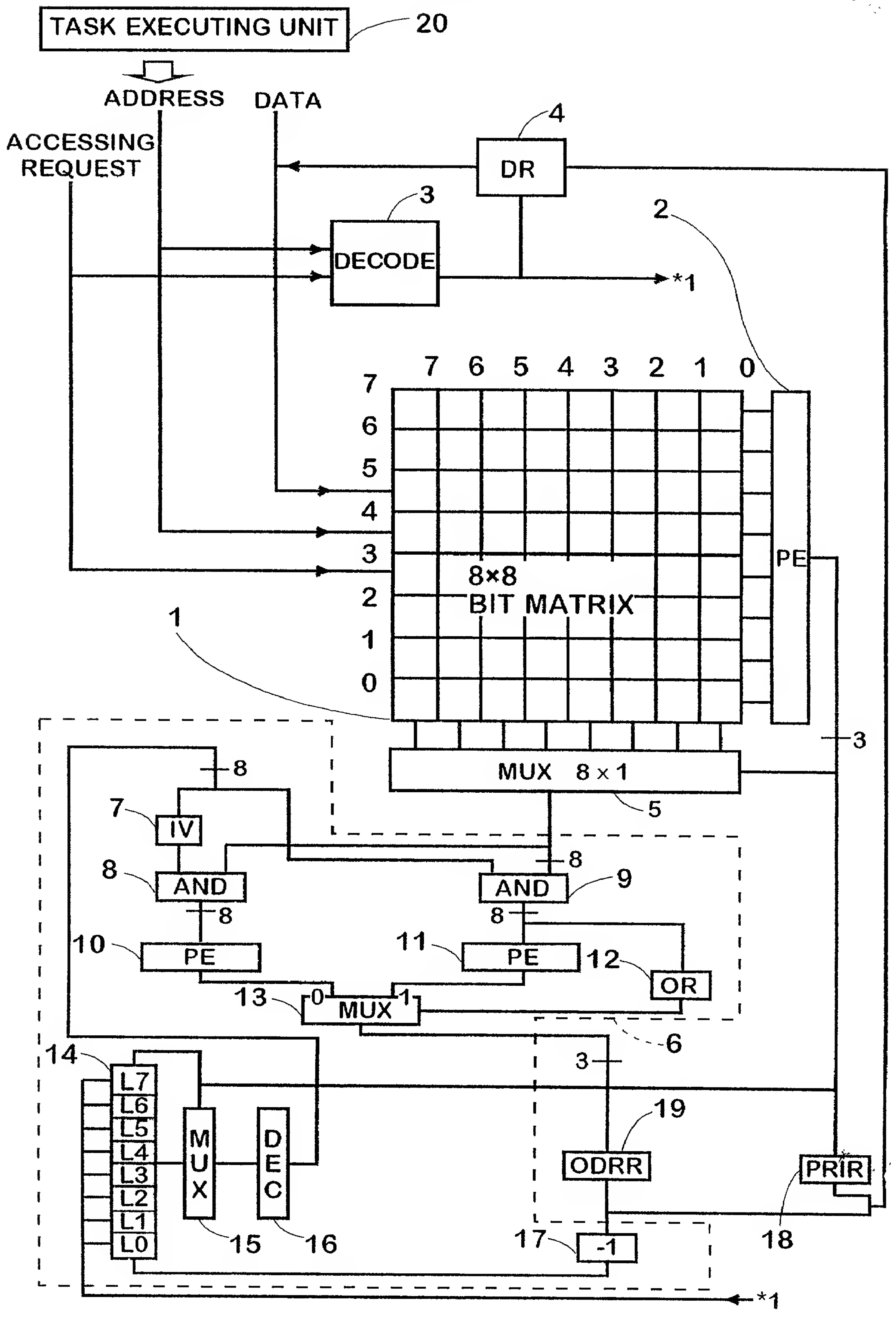
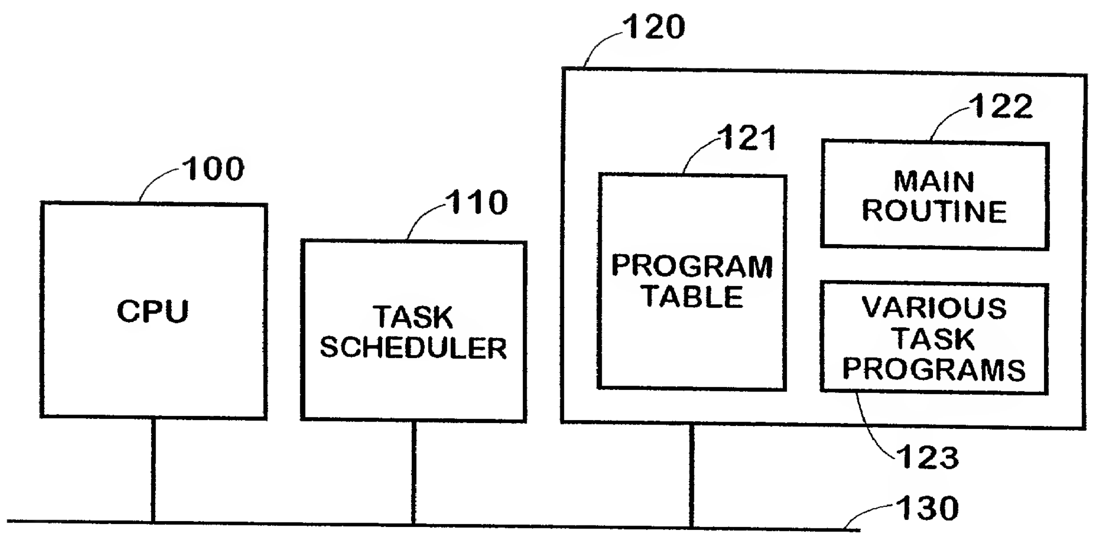
#3

Fig.1



TASK EXECUTING UNIT
20
ADDRESS
DATA
ACCESSING REQUEST
4
DR
3
DECODE
2
*1
7 6 5 4 3 2 1 0
7
6
5
4
3
2
1
0
8×8 BIT MATRIX
PE
1
MUX 8×1
5
3
8
7
IV
8
AND
AND
9
8
8
10
PE
11
PE
12
OR
13
0
MUX
1
6
14
L7
L6
L5
L4
L3
L2
L1
L0
MUX
DEC
15
16
3
19
ODRR
PRIR
17
-1
18
*1

Fig.2



120
122
121
100
110
MAIN ROUTINE
PROGRAM TABLE
CPU
TASK SCHEDULER
VARIOUS TASK PROGRAMS
123
130

Fig.3

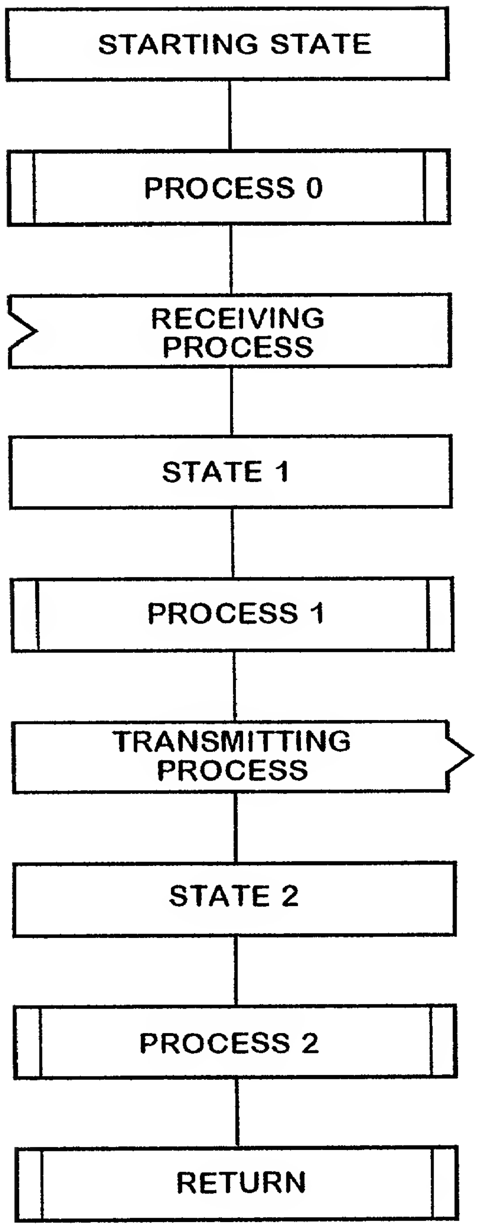

STARTING STATE
PROCESS 0
RECEIVING PROCESS
STATE 1
PROCESS 1
TRANSMITTING PROCESS
STATE 2
PROCESS 2
RETURN

**Fig.4**

```
func()
{
    proc0:
        Process Contents 0;
    recieve(chanel0, data);
    proc1:
        Process Contents 1;
    send(chanel0, data);
    proc2:
        Process Contents 2;
    return;
}
```

Fig.5A

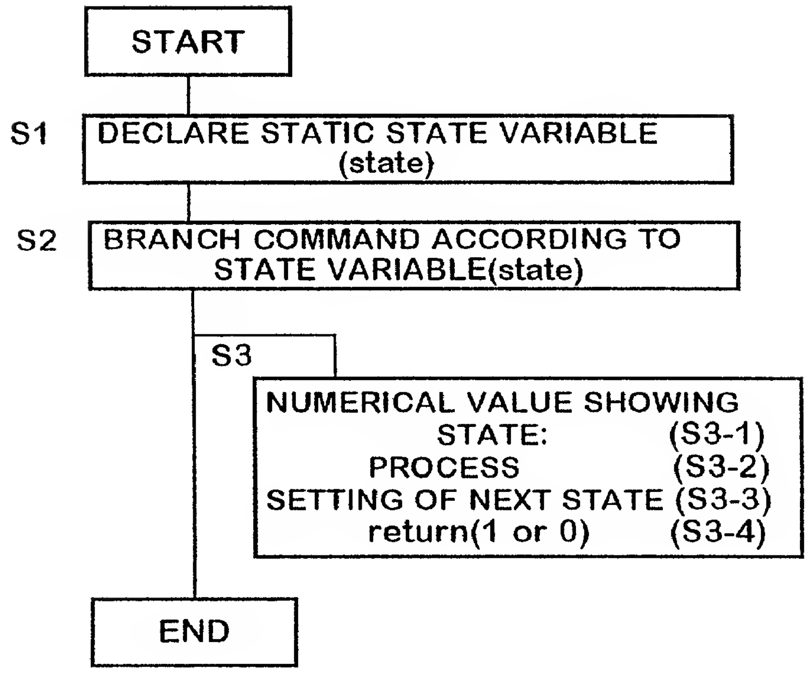

START
S1
DECLARE STATIC STATE VARIABLE (state)
S2
BRANCH COMMAND ACCORDING TO STATE VARIABLE(state)
S3
NUMERICAL VALUE SHOWING STATE: (S3-1)
PROCESS (S3-2)
SETTING OF NEXT STATE (S3-3)
return(1 or 0) (S3-4)
END

**Fig.5B**

```
static int state;  // S1
func()
{
     switch(state&0x3) { // S2
     //S3
     case0:   //S3-1
             Process Contents 0;   //   S3-2
             state=1;      //   S3-3
             return(0);  //    S3-4
     case1:
             get(chanel0,data);
             Process Contents 1;
             send(chanel0, data);
             state=2;
             return(0);
     case2:
             Process Contents 2;
             state=0;
             return(0);
     defaults:
             state=0;
             return(0);
    }
}
```

Fig.6A

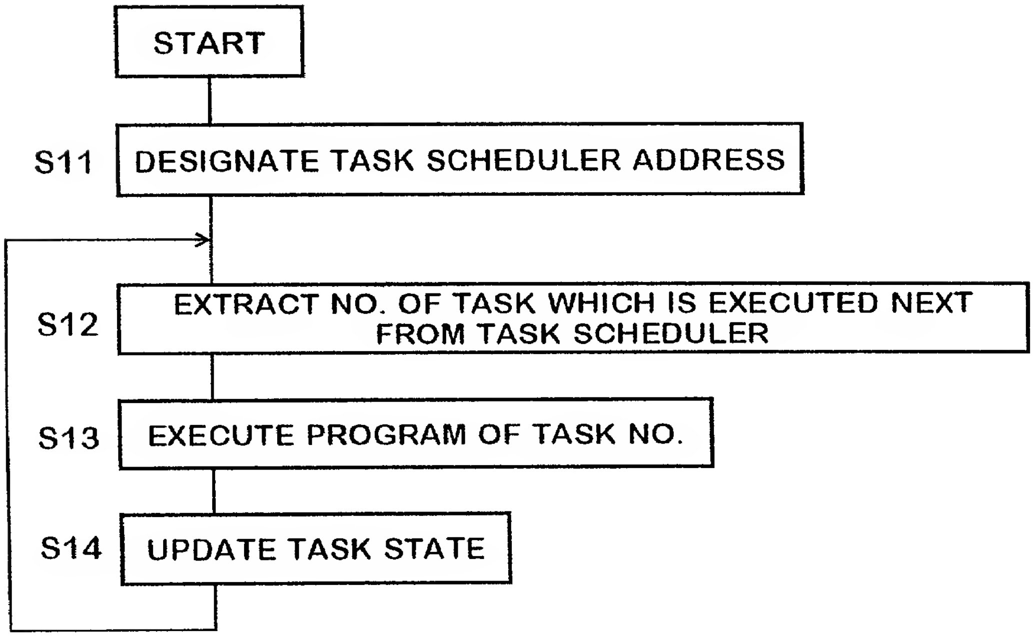

START
S11
DESIGNATE TASK SCHEDULER ADDRESS
S12
EXTRACT NO. OF TASK WHICH IS EXECUTED NEXT FROM TASK SCHEDULER
S13
EXECUTE PROGRAM OF TASK NO.
S14
UPDATE TASK STATE

Fig.6B

```
int. *sp;
sp= TASK_SCHEDULER_ADR;  // S11

while(1) {
     NUM= *sp;                    // S12
     state[NUM]=func[NUM]();    // S13&S14
     }
```

Fig.7A

task0 b110_101
task1 b011_110
task2 b011_011

ADDRESS OF EACH TASK

Fig.7B

$$\begin{array}{c|cccccccc}
 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

SITUATION OF 8 × 8 BIT MATRIX

Fig.8A

```
static int sta0;

int task0()
{
    switch(sta0&0x1) {
    case0:                //13_S
            sta0=1;
            return(0);
    case1:                //30_S
            sta0=0;
            return(0);
    }
}
```

CONTENS OF TASK 0

Fig.8B

```
static int sta1;

int task1()
{
    switch(sta1&0x1) {
    case0:                //19_S
            sta1=1;
            return(1);
    case1:
            sta1=0;
            return(0);
    }
}
```

CONTENS OF TASK 1

Fig.8C

```
static int sta2;

int task2()
{
    switch(sta2&0x1) {
    case0:                //26_S
            state(task0_id)=1;
            sta2=1;
            return(1);
    case1:
            sta2=0;
            return(0);
    }
}
```

CONTENS OF TASK 2

Fig.9

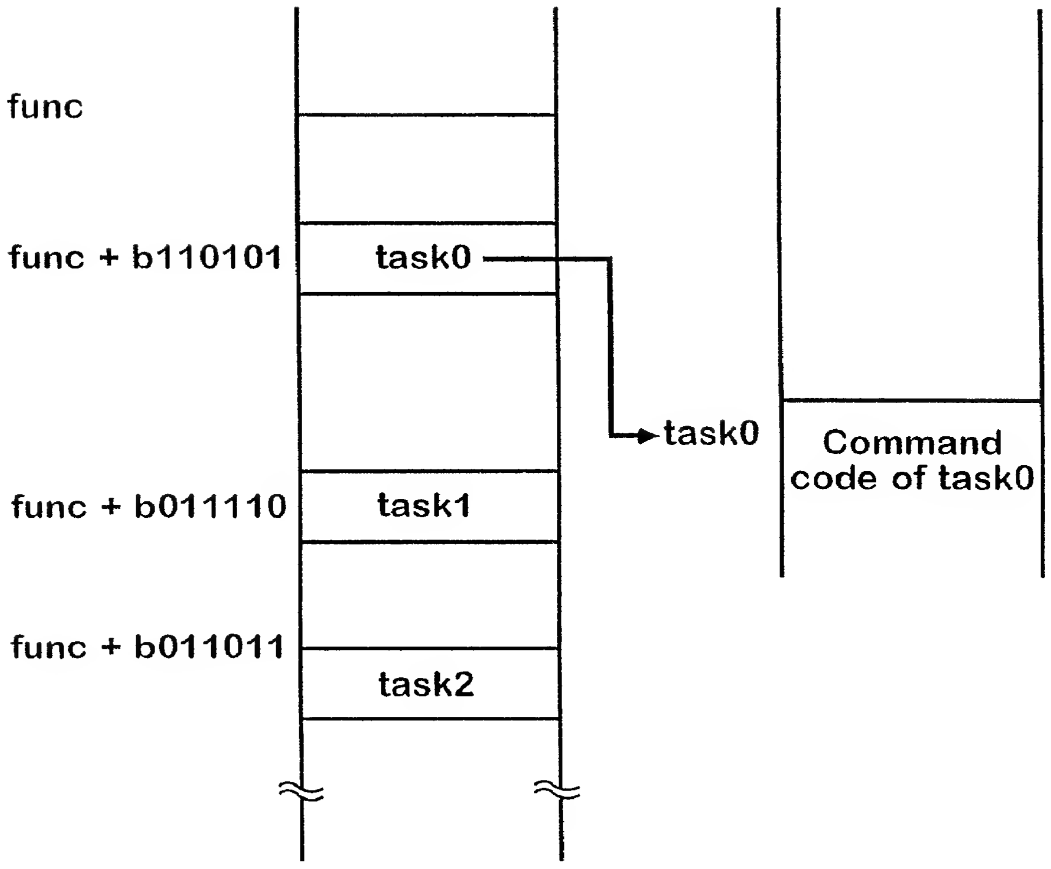
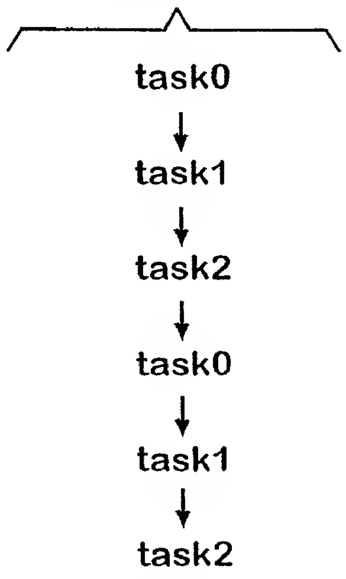
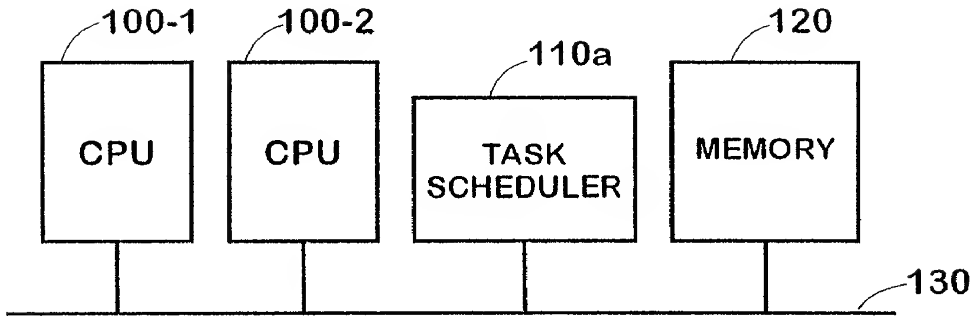

func
func + b110101
task0
task0
Command code of task0
func + b011110
task1
func + b011011
task2

Fig.10


task0
task1
task2
task0
task1
task2

## Fig.11


100-1
100-2
110a
120
CPU
CPU
TASK SCHEDULER
MEMORY
130

## Fig.12

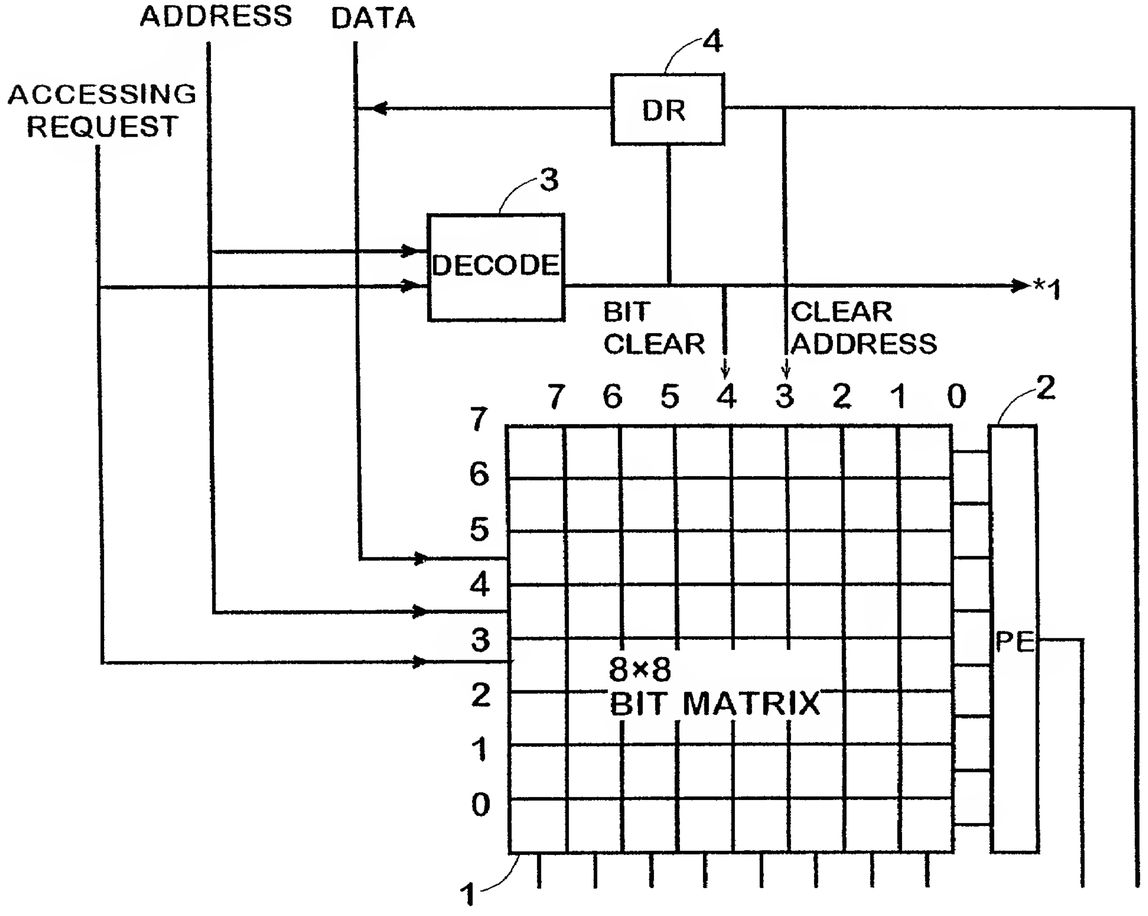
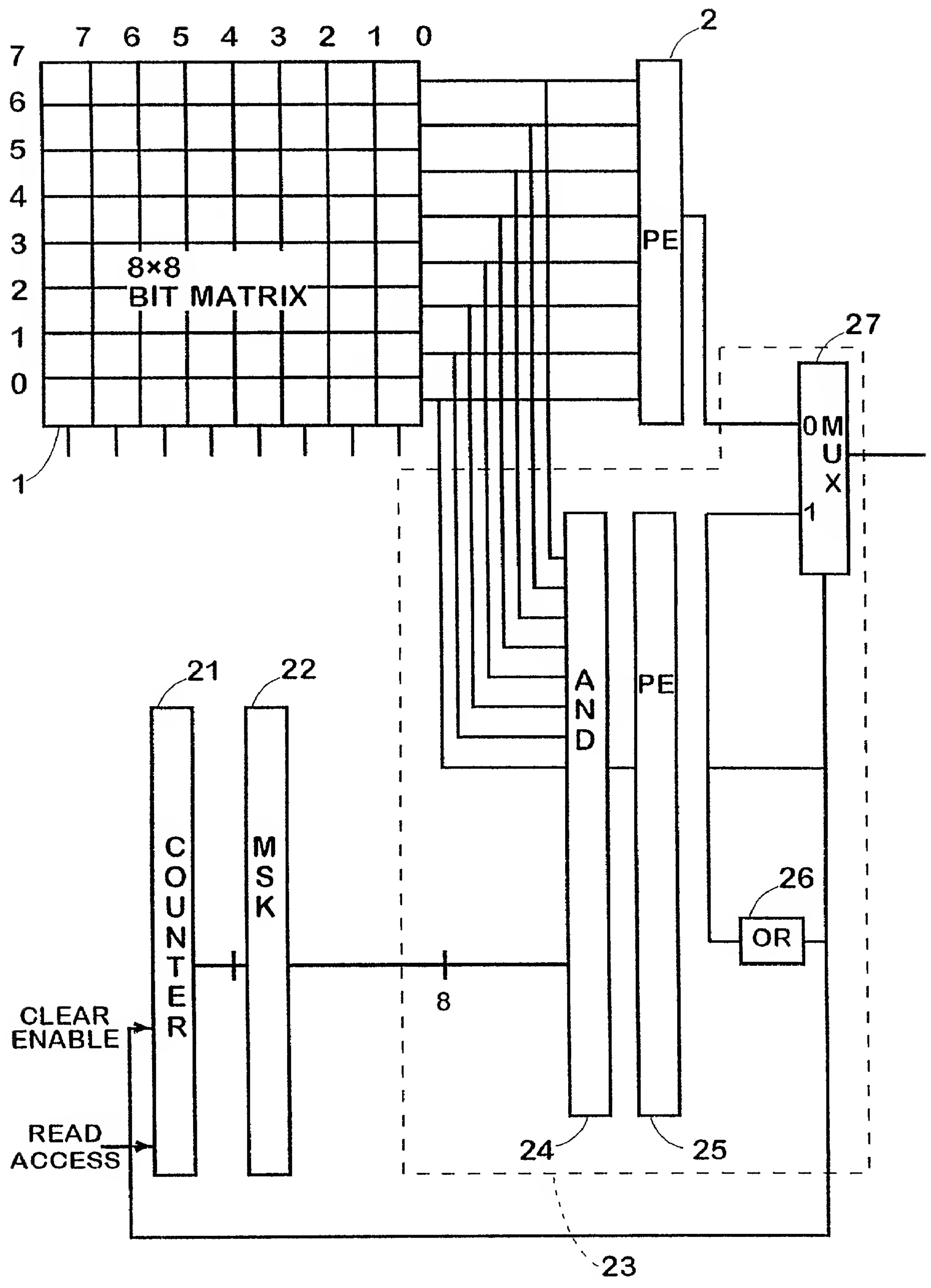
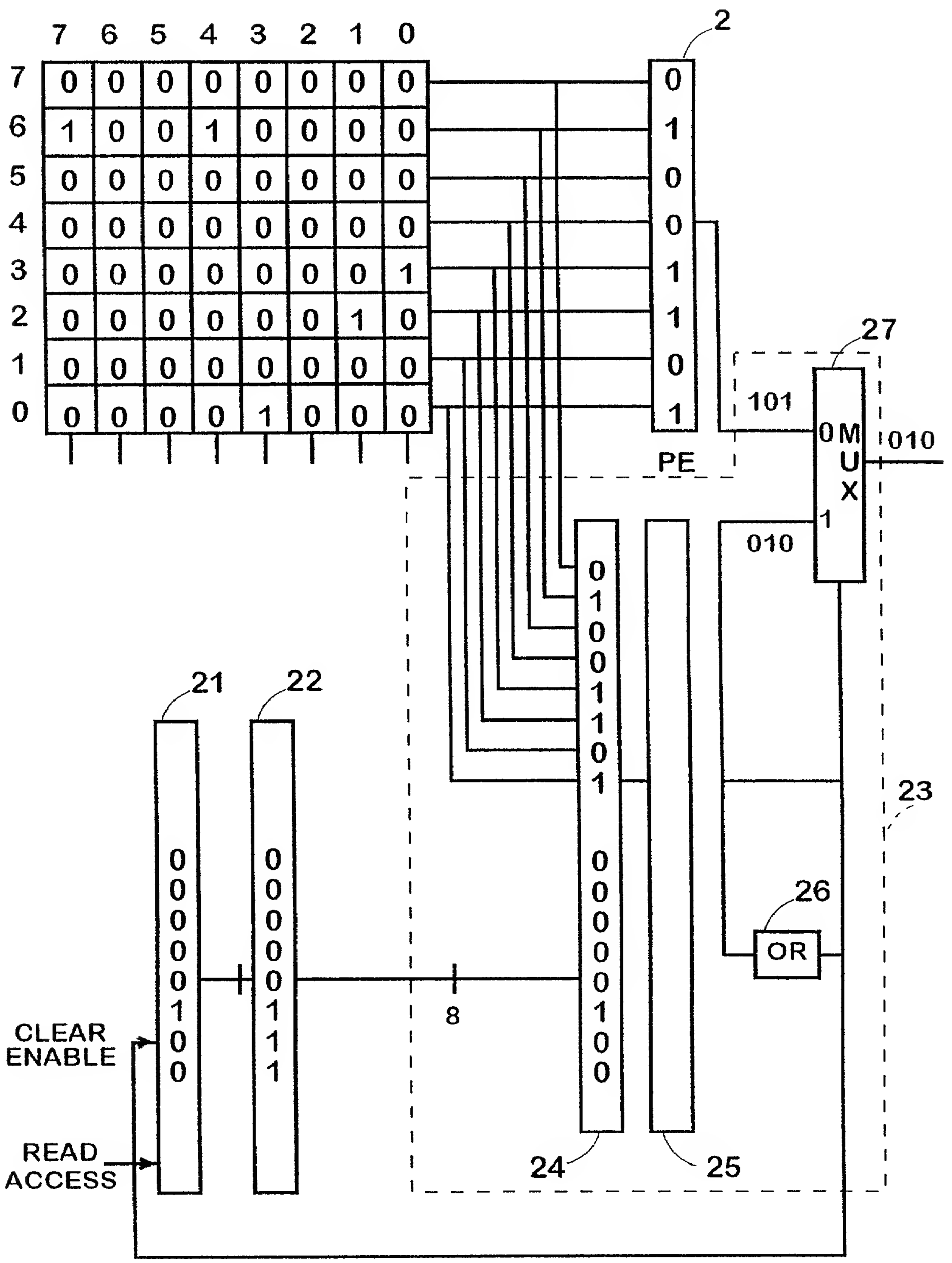

ADDRESS
DATA
ACCESSING REQUEST
4
DR
3
DECODE
*1
BIT CLEAR
CLEAR ADDRESS
7 6 5 4 3 2 1 0
2
7
6
5
4
3
2
1
0
8×8 BIT MATRIX
PE
1

Fig.13



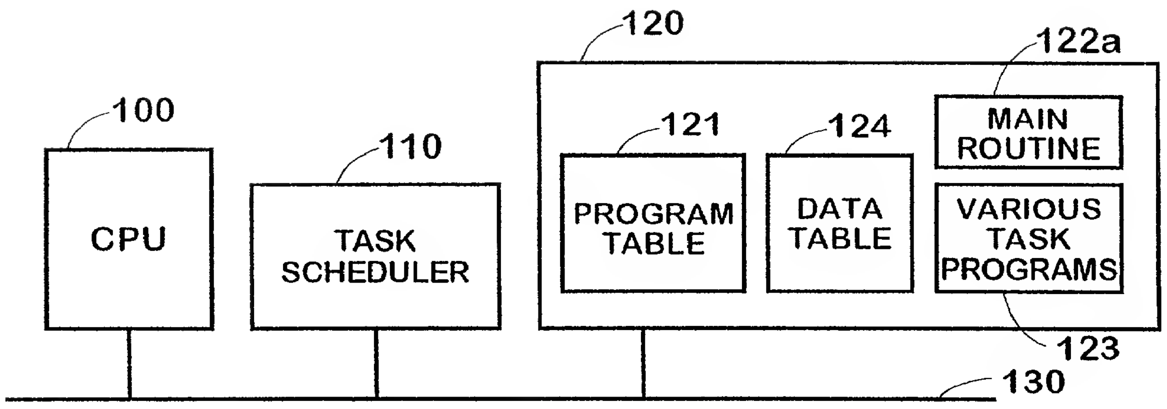
7 6 5 4 3 2 1 0
7
6
5
4
3
2
1
0
8×8
BIT MATRIX
1
2
PE
27
0
MUX
1
AND
PE
21
22
COUNTER
MSK
8
26
OR
CLEAR ENABLE
READ ACCESS
24
25
23

**Fig.14**



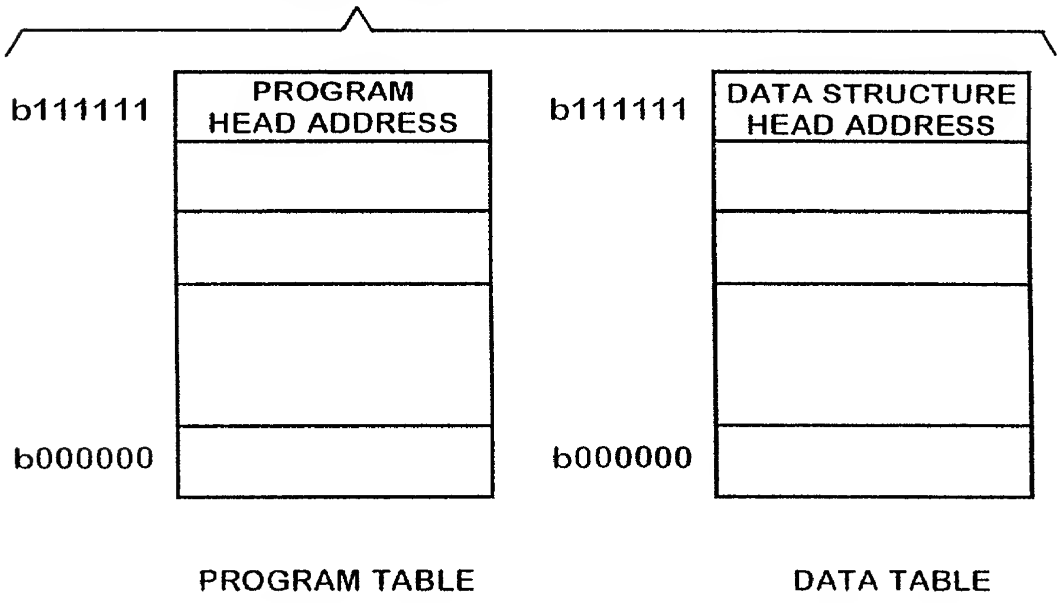
7 6 5 4 3 2 1 0
7 0 0 0 0 0 0 0 0
6 1 0 0 1 0 0 0 0
5 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0
3 0 0 0 0 0 0 0 1
2 0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0
2
0 1 0 0 1 1 0 1
PE
101
27
0
MUX
1
010
010
0 1 0 0 1 1 0 1
0 0 0 0 0 1 0 0
21
22
0 0 0 0 0 1 0 0
0 0 0 0 0 1 1 1
8
CLEAR ENABLE
READ ACCESS
23
26
OR
24
25

Fig.15



100
CPU
110
TASK SCHEDULER
120
121
PROGRAM TABLE
124
DATA TABLE
122a
MAIN ROUTINE
VARIOUS TASK PROGRAMS
123
130

Fig.16



b111111
PROGRAM HEAD ADDRESS
b000000
PROGRAM TABLE
b111111
DATA STRUCTURE HEAD ADDRESS
b000000
DATA TABLE

Fig.17A

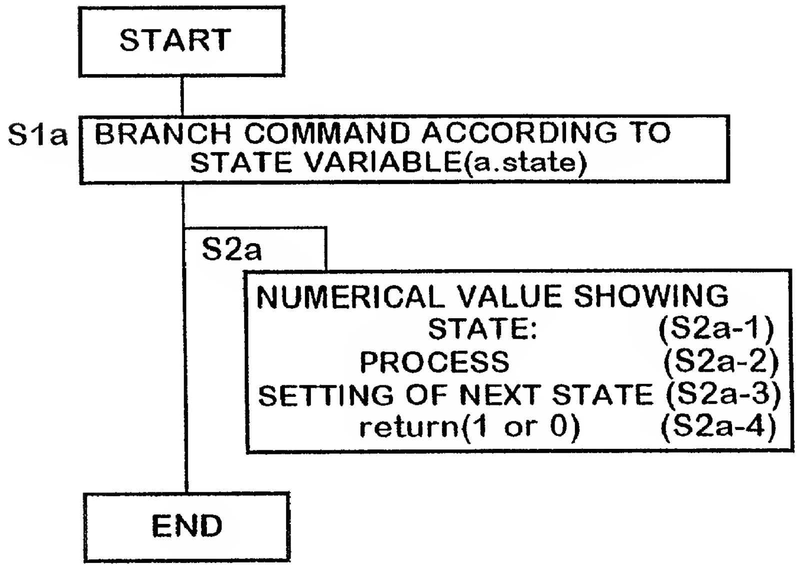


Fig.17B

```
int.  func(struct xxx*a)
{
      switch(a.state&0x3) { // S1a
      //S2a
      case0:   //S2a-1
              Process contents 0;   //   S2a-2
              state=1;      //   S2a-3
              return(0);  //    S2a-4
      case1:
              get(chanel0,data);
              Process contents 1;
              send(chanel0, data);
              a.state=2,
              return(0);
      case2:
             Process contents 2;
             a.state=0;
              return(0);
      defaults:
              a.state=0;
              return(0);
     }
}
```

**Fig. 18A**

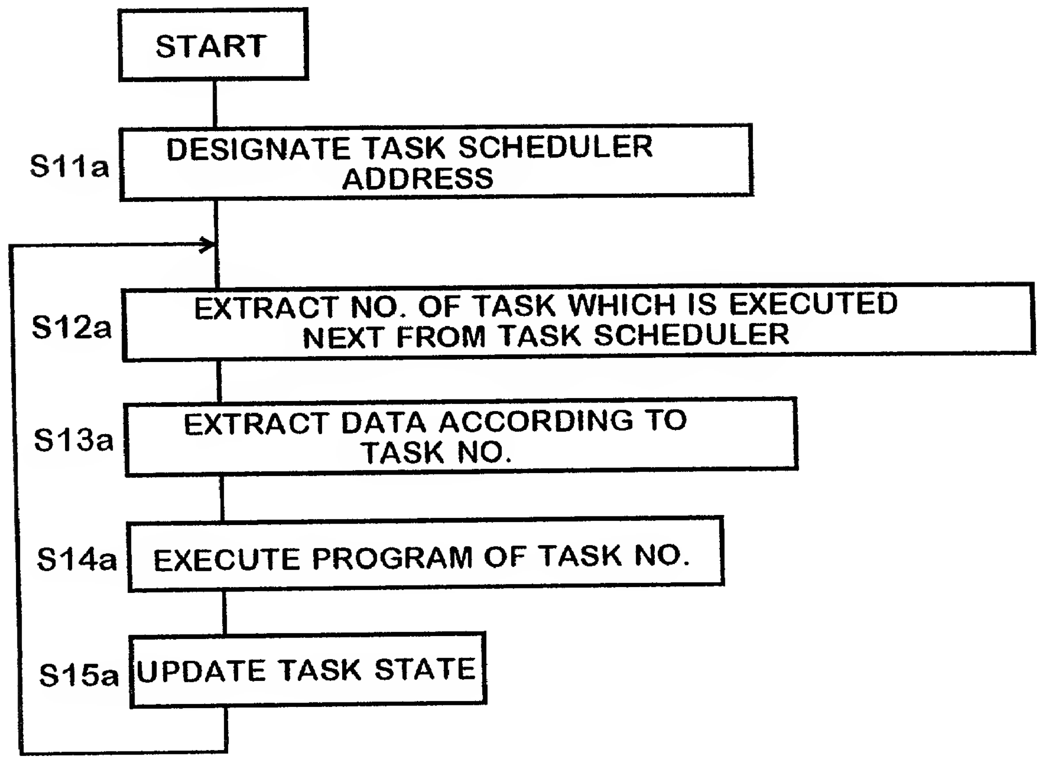
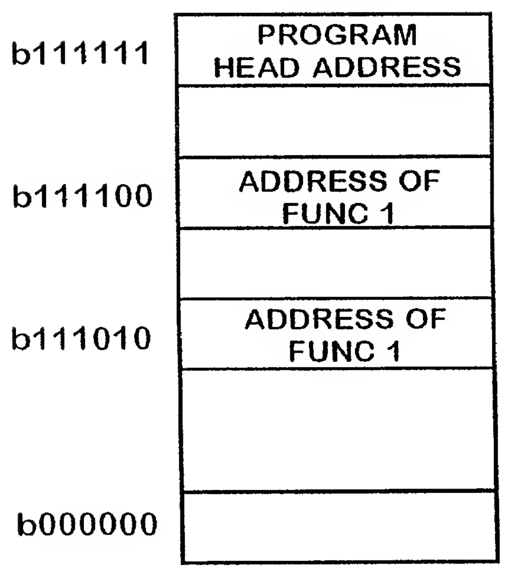

START
S11a
DESIGNATE TASK SCHEDULER ADDRESS
S12a
EXTRACT NO. OF TASK WHICH IS EXECUTED NEXT FROM TASK SCHEDULER
S13a
EXTRACT DATA ACCORDING TO TASK NO.
S14a
EXECUTE PROGRAM OF TASK NO.
S15a
UPDATE TASK STATE

**Fig. 18B**

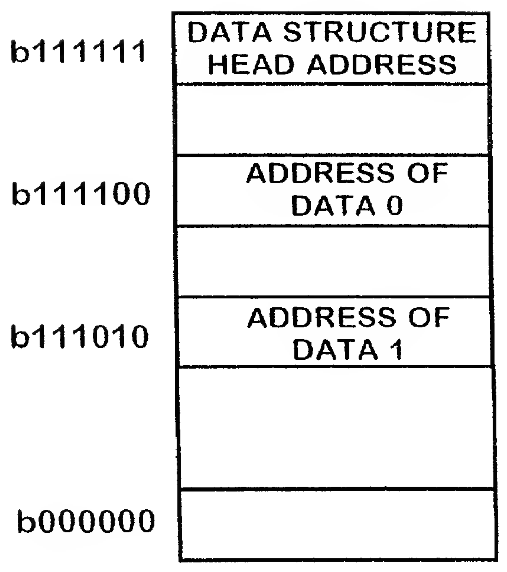
```
int. *sp;
int. *data[];
sp= TASK_SCHEDULER_ADR;  // S11a

while(1) {
     NUM= *sp;                        // S12a
     state[NUM]=func[NUM](data[NUM]);  // S14a&15a&S13a
     }
```

Fig.19A



PROGRAM TABLE

Fig.19B



DATA TABLE